Laravel : Formulaires HTTP et Relations Eloquent

Maîtriser les opérations CRUD et les relations de base de données

Formation Développement Digital

ISTA Tinghir - OFPPT

January 15, 2026

Plan du cours

Objectifs de cette section

- Comprendre les méthodes HTTP (POST, PUT, DELETE)
- Créer des formulaires de création (POST)
- Créer des formulaires de modification (PUT)
- Implémenter la suppression (DELETE)
- Gérer la validation des données

Créer un Modèle avec Migration et Contrôleur

Commande complète (Recommandée)

Créer un modèle avec migration, contrôleur ressource et factory :

```
php artisan make:model Product -mcr
```

- **-m** : Crée la migration
- **-c** : Crée le contrôleur
- **-r** : Le contrôleur sera de type ressource

Résultat

Cette commande crée 3 fichiers :

- app/Models/Product.php
- app/Http/Controllers/ProductController.php (ressource)
- database/migrations/XXXX_create_products_table.php

Autres Commandes Artisan

Créer uniquement un contrôleur ressource

```
php artisan make:controller ProductController --resource
# OU (version courte)
php artisan make:controller ProductController -r
```

Créer un modèle avec différentes options

```
# Avec migration seulement
php artisan make:model Product -m

# Avec migration et factory
php artisan make:model Product -mf

# Avec migration, factory et seeder
php artisan make:model Product -mfs
```

Options des Commandes Artisan

Option	Description
-m	Crée une migration
-c	Crée un contrôleur
-r	Contrôleur de type ressource
-f	Crée un factory
-s	Crée un seeder
-a	Crée tout (migration, factory, seeder, controller)

Exemple pratique

```
php artisan make:model Article -mcr
```

Crée le modèle Article avec sa migration et son contrôleur ressource

Routes Ressource

Route::resource()

Laravel offre une méthode pratique pour créer toutes les routes CRUD :

```php
// routes/web.php
use App\Http\Controllers\UserController;

Route::resource('users', UserController::class);
```

Equivalent

Cette ligne génère automatiquement 7 routes CRUD

Les 7 Routes Générées

1. GET /users (index) - Liste des utilisateurs
2. GET /users/create (create) - Formulaire de création
3. POST /users (store) - Enregistrer un utilisateur
4. GET /users/{user} (show) - Afficher un utilisateur
5. GET /users/{user}/edit (edit) - Formulaire de modification
6. PUT/PATCH /users/{user} (update) - Mettre à jour
7. DELETE /users/{user} (destroy) - Supprimer

Vue : Formulaire de Création

```blade
<form action="{{ route('users.store') }}" method="POST">
    @csrf

    <div class="form-group">
        <label for="name">Nom</label>
        <input type="text"
               class="form-control"
               id="name"
               name="name"
               value="{{ old('name') }}"
               required>
        @error('name')
            <div class="text-danger">{{ $message }}</div>
        @enderror
    </div>

    <div class="form-group">
        <label for="email">Email</label>
        <input type="email"
               class="form-control"
               id="email"
               name="email"
               required>
    </div>

    <button type="submit">Cr er</button>
</form>
```

Contrôleur : Méthode Store

```php
public function store(Request $request)
{
    // Validation des donn es
    $validated = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email',
        'password' => 'required|min:8|confirmed',
        'role_id' => 'required|exists:roles,id'
    ]);

    // Cr ation de l'utilisateur
    $user = User::create([
        'name' => $validated['name'],
        'email' => $validated['email'],
        'password' => Hash::make($validated['password']),
        'role_id' => $validated['role_id']
    ]);

    // Cr er le profil automatiquement
    $user->profile()->create([
        'bio' => '',
        'phone' => ''
    ]);

    return redirect()->route('users.index')
        ->with('success', 'Utilisateur cr    avec succ s');
}
```

Points Importants - POST

@csrf

Obligatoire ! Protège contre les attaques CSRF

old('field')

Récupère les anciennes valeurs en cas d'erreur de validation

@error('field')

Affiche les erreurs de validation pour un champ spécifique

Validation

La validation s'effectue avec request-¿validate()

Vue : Formulaire de Modification

```blade
<form action="{{ route('users.update', $user) }}" method="POST">
    @csrf
    @method('PUT')

    <div class="form-group">
        <label for="name">Nom</label>
        <input type="text"
               class="form-control"
               id="name"
               name="name"
               value="{{ old('name', $user->name) }}"
               required>
    </div>

    <div class="form-group">
        <label for="password">Nouveau mot de passe</label>
        <input type="password"
               class="form-control"
               id="password"
               name="password">
        <small>Laisser vide pour ne pas changer</small>
    </div>

    <button type="submit">Mettre    jour</button>
</form>
```

Contrôleur : Méthode Update

```php
public function update(Request $request, User $user)
{
    $validated = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email,' . $user->id,
        'password' => 'nullable|min:8|confirmed',
        'role_id' => 'required|exists:roles,id'
    ]);

    $user->name = $validated['name'];
    $user->email = $validated['email'];
    $user->role_id = $validated['role_id'];

    // Mettre    jour le mot de passe seulement s'il est fourni
    if (!empty($validated['password'])) {
        $user->password = Hash::make($validated['password']);
    }

    $user->save();

    return redirect()->route('users.index')
        ->with('success', 'Utilisateur modifi ');
}
```

Points Importants - PUT

@method('PUT')

HTML ne supporte que GET et POST. Laravel utilise un champ caché pour simuler PUT

old('field', $default)

Le deuxième paramètre définit la valeur par défaut

Validation unique

unique:users,email,$user-¿id exclut l'utilisateur actuel

Mot de passe optionnel

Utilisez nullable dans la validation

Vue : Bouton de Suppression

```
<table class="table">
    <thead>
        <tr>
            <th>Nom</th>
            <th>Email</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        @foreach($users as $user)
        <tr>
            <td>{{ $user->name }}</td>
            <td>{{ $user->email }}</td>
            <td>
                <form action="{{ route('users.destroy', $user) }}"
                      method="POST"
                      style="display:inline;"
                      onsubmit="return confirm('Confirmer ?')">
                    @csrf
                    @method('DELETE')
                    <button type="submit" class="btn btn-danger">
                        Supprimer
                    </button>
                </form>
            </td>
        </tr>
        @endforeach
    </tbody>
</table>
```

Contrôleur : Méthode Destroy

```php
public function destroy(User $user)
{
    try {
        // Supprimer les relations
        $user->profile()->delete();
        $user->permissions()->detach();

        // Supprimer l'utilisateur
        $user->delete();

        return redirect()->route('users.index')
            ->with('success', 'Utilisateur supprim ');

    } catch (\Exception $e) {
        return redirect()->route('users.index')
            ->with('error', 'Erreur : ' . $e->getMessage());
    }
```

Points Importants - DELETE

Confirmation

Toujours demander confirmation : onsubmit="return confirm(...)"

Gestion des relations

Avant de supprimer, pensez aux relations :

- Détacher les relations Many-to-Many
- Supprimer les relations One-to-One/One-to-Many
- Ou configurer onDelete('cascade')