TP4 / CONSTRUIRE UN CRUD COMPLET AVEC LARAVEL

🎯 Objectif du TP

Notre application sait maintenant **lire** des clients depuis SQLite, la base de données.

Mais une véritable application doit permettre de :

```
Créer
Lire
Modifier
Supprimer
```

C'est le **CRUD** :

```
C – Create
R – Read
U – Update
D – Delete
```

À la fin du TP, nous disposerons d'une véritable gestion de clients.

1. Le CRUD

Pour notre ressource `Client`, nous voulons pouvoir :

Action	HTTP	URL
Lister	GET	`/clients`
Afficher le formulaire	GET	`/clients/create`
Enregistrer	POST	`/clients`
Afficher un client	GET	`/clients/{client}`
Afficher modification	GET	`/clients/{client}/edit`
Modifier	PUT/PATCH	`/clients/{client}`
Supprimer	DELETE	`/clients/{client}`

2. Le Resource Controller

Laravel fournit un mécanisme permettant de construire rapidement les contrôleurs destinés à gérer une ressource.

Nous allons supprimer l'ancien Controller si nécessaire et créer :

```
php artisan make:controller ClientController --resource
```

Le Controller contient alors les méthodes :

```
index()
create()
store()
show()
edit()
update()
destroy()
```

Associons chaque méthode à son rôle

Méthode	Rôle
`index()`	……
`create()`	……
`store()`	……
`show()`	……
`edit()`	……
`update()`	……
`destroy()`	……

3. Les routes Resource

Dans :

```
routes/web.php
```

utilisez :

```
use App\Http\Controllers\ClientController;

Route::resource('clients', ClientController::class);
```

Cette seule déclaration génère les routes CRUD correspondantes.

Pour les observer :

```
php artisan route:list
```

Repérez les routes :

```
clients.index
clients.create
clients.store
clients.show
clients.edit
clients.update
clients.destroy
```

Exercice

Associez chaque route à son Controller :

```
GET /clients  → .....................................

GET /clients/create  → ..............................

POST /clients  → ....................................

GET /clients/{client}  → ............................

GET /clients/{client}/edit  → .......................

PUT/PATCH /clients/{client}  → ......................

DELETE /clients/{client}  → .........................
```

4. Afficher la liste

Dans :

```
public function index()
```

récupérez tous les clients :

```
$clients = Client::all();

return view('clients.index', [
    'clients' => $clients
]);
```

Votre View devra afficher :

```
Nom
Email
Ville
Actions
```

Par exemple :

```
Jean Dupont     jean@example.com       Paris      Voir | Modifier | Supprimer
Marie Martin    marie@example.com      Lyon       Voir | Modifier | Supprimer
```

5. Ajouter le bouton « Ajouter un client »

Dans la View :

```
<a href="{{ route('clients.create') }}">
    Ajouter un client
</a>
```

Question

Pourquoi utilise-t-on :

```
route('clients.create')
```

plutôt qu'une URL écrite directement :

```
/clients/create
```

.....

6. Afficher le formulaire

Dans :

```php
public function create()
{
    return view('clients.create');
}
```

Créez :

```
resources/views/clients/create.blade.php
```

Utilisez :

```
@extends('layouts.app')

@section('content')

    <h2>Ajouter un client</h2>

    <form method="POST" action="{{ route('clients.store') }}">

        @csrf

        <label>Nom</label>
        <input type="text" name="name">

        <label>Email</label>
        <input type="email" name="email">

        <label>Ville</label>
        <input type="text" name="city">

        <button type="submit">
            Enregistrer
        </button>

    </form>

@endsection
```

⚠ À retenir

Un formulaire POST Laravel doit notamment comporter :

```
@csrf
```

Nous étudierons précisément la validation dans le prochain cours.

7. Enregistrer le nouveau client

Dans :

```
public function store(...)
```

récupérez les données envoyées par le formulaire et créez le client :

```
Client::create([
    'name' => $request->name,
    'email' => $request->email,
    'city' => $request->city,
]);
```

N’oubliez pas d’importer :

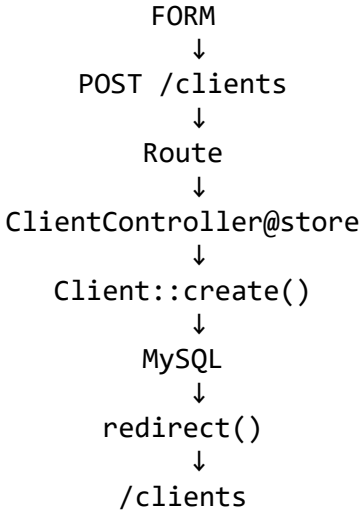
```
use Illuminate\Http\Request;
```

Puis redirigez vers la liste :

```
return redirect()->route('clients.index');
```

8. Comprendre le parcours du formulaire

Lorsque l’utilisateur clique sur **Enregistrer** :



Question

À quel moment les données sont-elles écrites dans la base ?

……

9. Afficher un client

Nous allons maintenant créer :

```
GET /clients/{client}
```

Dans le Controller :

```php
public function show(Client $client)
{
    return view('clients.show', [
        'client' => $client
    ]);
}
```

Laravel peut effectuer automatiquement le **route model binding** lorsque le type du paramètre correspond au Model attendu.

Créez :

```
resources/views/clients/show.blade.php
```

Affichez :

```
@extends('layouts.app')

@section('content')

    <h2>{{ $client->name }}</h2>

    <p>Email : {{ $client->email }}</p>

    <p>Ville : {{ $client->city }}</p>

    <a href="{{ route('clients.index') }}">
        Retour
    </a>

@endsection
```

10. Ajouter le bouton « Voir »

Dans `clients.index` :

```
<a href="{{ route('clients.show', $client) }}">
    Voir
</a>
```

Testez.

11. Modifier un client

Dans :

```
public function edit(Client $client)
{
    return view('clients.edit', [
        'client' => $client
    ]);
}
```

Créez :

```
resources/views/clients/edit.blade.php
```

Le formulaire devra être prérempli :

```
<input
    type="text"
    name="name"
    value="{{ $client->name }}"
>
```

Même principe pour :

```
email
city
```

Le formulaire :

```
<form method="POST"
      action="{{ route('clients.update', $client) }}">

    @csrf

    @method('PUT')

    ...

</form>
```

12. Pourquoi `@method('PUT')` ?

Les formulaires HTML classiques permettent notamment d'envoyer :

```
GET
POST
```

Laravel permet de représenter les autres verbes HTTP grâce au **method spoofing** dans les formulaires.

Ainsi :

```
@method('PUT')
```

indique que nous souhaitons effectuer une requête de mise à jour.

13. Mettre à jour le client

Dans :

```
public function update(Request $request, Client $client)
```

faites :

```
$client->update([
    'name' => $request->name,
    'email' => $request->email,
    'city' => $request->city,
]);
```

Puis :

```
return redirect()->route('clients.index');
```

Testez :

1. ouvrir un client ;
2. cliquer sur Modifier ;
3. modifier la ville ;
4. enregistrer ;
5. vérifier la base de données.

14. Supprimer un client

Dans la liste, ajoutez :

```
<form method="POST"
      action="{{ route('clients.destroy', $client) }}">

    @csrf

    @method('DELETE')
```

```
    <button type="submit">
        Supprimer
    </button>

</form>
```

Dans le Controller :

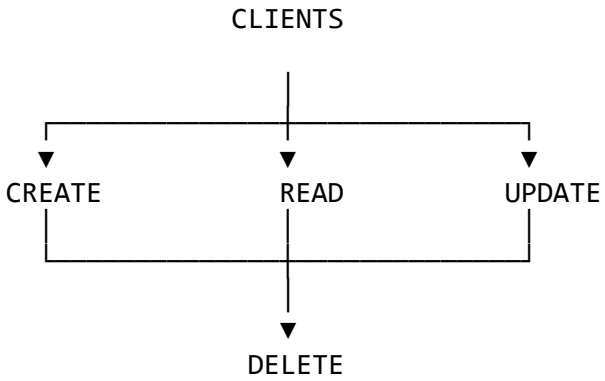
```php
public function destroy(Client $client)
{
    $client->delete();

    return redirect()->route('clients.index');
}
```

Testez.

15. Notre CRUD est terminé

Nous possédons maintenant :



Notre application sait maintenant :

- ✅ créer un client
- ✅ afficher les clients
- ✅ afficher un client
- ✅ modifier un client
- ✅ supprimer un client

16. Exercice — Améliorer la liste

Dans la liste des clients, ajoutez :

```
Nombre total de clients : X
```

Vous pouvez utiliser :

```
$clients->count()
```

Puis ajoutez une information :

```
Client créé le : ...
```

en utilisant :

```
$client->created_at
```

17. Exercice — Ajouter une navigation complète

Dans votre Layout :

```
Accueil
Clients
Produits
Contact
```

Le lien **Clients** doit pointer vers :

```
route('clients.index')
```

Le lien **Ajouter un client** doit pointer vers :

```
route('clients.create')
```

☆ Défi final — CRUD Produits

Vous allez maintenant réaliser **sans recopier étape par étape** un CRUD complet pour `Product`.

Le Model et la table ont été créés au TP3.

Vous devez réaliser :

```
ProductController
    ↓
Route::resource()
    ↓
index()
create()
store()
show()
edit()
update()
destroy()
```

avec les champs :

```
name
price
stock
```

Votre application doit permettre :

```
/products
/products/create
/products/{product}
/products/{product}/edit
```

et les opérations :

```
Créer
Afficher
Modifier
Supprimer
```

☆☆ Défi supplémentaire

Dans la liste des produits :

Si :

`stock < 5`

affichez :

`⚠ Stock faible`

Sinon :

`Disponible`

➢BILAN DU TP4

Vous devez maintenant savoir :

☐ comprendre le CRUD

☐ créer un Resource Controller

☐ utiliser `Route::resource()`

☐ utiliser `route()`

☐ créer un formulaire POST

☐ utiliser `@csrf`

☐ récupérer les données d'un formulaire

☐ créer un enregistrement Eloquent

☐ afficher un enregistrement

☐ modifier un enregistrement

☐ supprimer un enregistrement

☐ utiliser `@method('PUT')`

☐ utiliser `@method('DELETE')`

☐ comprendre le Route Model Binding

➢ QUESTION FINALE

Sans regarder votre code, expliquez ce qui se passe lorsque l'utilisateur :

1. ouvre `/clients` ;
2. clique sur « Ajouter un client » ;
3. remplit le formulaire ;
4. clique sur « Enregistrer ».

Votre réponse doit utiliser les mots :

```
Route
Controller
Request
Eloquent
Database
Redirect
View
```

……

……

……

……

TP4 - Reponses completees

Annexe ajoutee apres le sujet original

Le PDF du professeur ne contient pas de champs a remplir. Le sujet est preserve et cette annexe repond aux questions.

Page 2 - Role des methodes du Resource Controller

Methode	Role
index()	Lister tous les clients.
create()	Afficher le formulaire de creation.
store()	Recevoir le formulaire et creer le client dans la base.
show()	Afficher le detail d un client.
edit()	Afficher le formulaire de modification pre-rempli.
update()	Enregistrer les modifications du client.
destroy()	Supprimer le client.

Page 3 - Routes et Controller

Route	Action
GET /clients	ClientController@index
GET /clients/create	ClientController@create
POST /clients	ClientController@store
GET /clients/{client}	ClientController@show
GET /clients/{client}/edit	ClientController@edit
PUT/PATCH /clients/{client}	ClientController@update
DELETE /clients/{client}	ClientController@destroy

Page 4 - Pourquoi utiliser route('clients.create') ?

Cette fonction utilise le nom officiel de la route plutot qu une URL ecrite en dur. Si l URL change, Laravel met a jour le lien partout ou le nom est utilise. Le code reste donc lisible et facile a maintenir.

Page 6 - Quand les donnees sont-elles ecrites ?

Les donnees sont ecrites pendant Client::create(...) dans store(), apres reception du POST et avant la redirection.

Page 14 - Reponse finale

Quand l utilisateur ouvre **/clients**, la **Route** resource envoie la requete au **Controller** ClientController@index. Le Controller utilise **Eloquent** avec Client::all() pour lire depuis la **Database**, puis retourne la **View** clients.index.

Le clic Ajouter un client utilise la Route clients.create et retourne la View du formulaire. Au clic Enregistrer, le navigateur envoie une **Request** POST vers clients.store. Le Controller valide la Request, Eloquent execute Client::create() dans la Database, puis le Controller retourne un **Redirect** vers clients.index pour afficher la liste a jour.

Exercices et defis realises

• La liste Clients affiche le total et la date de creation.

• Le layout utilise les routes nommees clients.index et clients.create.

• Le CRUD Product est complet et Stock faible apparait lorsque stock est inferieur a 5.